Introduzione Assignment

Amanjot Singh matr. 152792
Leonardo Temperanza matr. 152831

1. CFG Fibonacci

La IR nella seguente immagine rappresenta la definizione della funzione printf. Tale funzione ha control flow lineare e dunque contiene soltanto un Basic Block (il blocco entry)

```
; Function Attrs: nounwind uwtable
define dso_local i32 @printf(i8* nocapture readonly %0, ...) local_unnamed_addr #0 {
  %2 = alloca [1 x %struct.__va_list_tag], align 16
  %3 = bitcast [1 x %struct.__va_list_tag]* %2 to i8*
  call void @llvm.lifetime.start.p0i8(i64 24, i8* nonnull %3) #2
  %4 = getelementptr inbounds [1 x %struct.__va_list_tag], [1 x %struct.__va_list_tag]* %2, i64 0, i64 0
  call void @llvm.va_start(i8* nonnull %3)
  %5 = load %struct._IO_FILE*, %struct._IO_FILE** @stdout, align 8, !tbaa !2
  %6 = call i32 @vfprintf(%struct._IO_FILE* %5, i8* %0, %struct.__va_list_tag* nonnull %4)
  call void @llvm.va_end(i8* nonnull %3)
  call void @llvm.lifetime.end.p0i8(i64 24, i8* nonnull %3) #2
  ret i32 %6
}
```

La funzione d'interesse Fibonacci invece presenta il seguente CFG:

Arco	Tipo	Condizione
Entry -> Basic Block 2	Fallthrough	
Basic Block 2 -> Basic Block 11	True	Se il registro %4 è uguale a 0
Basic Block 2 -> Basic Block 12	True	Se il registro %4 è uguale a 1
Basic Block 2 -> Basic Block 5	False	Altrimenti
Basic Block 5 -> Basic Block 2	True	
Basic Block 11 -> Basic Block 12	Fallthrough	

```
; Function Attrs: nounwind uwtable
define dso_local i32 @Fibonacci(i32 %0) local_unnamed_addr #0 {
  br label %2

2:                                                ; preds = %5, %1
  %3 = phi i32 [ 0, %1 ], [ %10, %5 ]
  %4 = phi i32 [ %0, %1 ], [ %7, %5 ]
  switch i32 %4, label %5 [
    i32 0, label %11
    i32 1, labSel %12
  ]

5:                                                ; preds = %2
  %6 = add nsw i32 %4, -1
  %7 = add nsw i32 %4, -2
  %8 = tail call i32 (i8*, ...) @printf(i8* nonnull dereferenceable(1) getelementptr inbounds ([22 x i8], [22 x i8]* @.str.2, i64 0, i64 0), i32 %4, i32 %6, i32 %7)
  %9 = tail call i32 @Fibonacci(i32 %6)
  %10 = add nsw i32 %9, %3
  br label %2

11:                                               ; preds = %2
  br label %12

12:                                               ; preds = %2, %11
  %13 = phi i8* [ getelementptr inbounds ([9 x i8], [9 x i8]* @.str, i64 0, i64 0), %11 ], [ getelementptr inbounds ([9 x i8], [9 x i8]* @.str.1, i64 0, i64 0), %2 ]
  %14 = tail call i32 (i8*, ...) @printf(i8* nonnull dereferenceable(1) %13)
  %15 = add nsw i32 %4, %3
  ret i32 %15
}
```

La rappresentazione intermedia di LLVM vede le chiamate a funzione come singole istruzioni che non alterano il controllo di flusso, per quanto riguarda la definizione dei Basic Block.

Pertanto LLVM non ritiene necessario creare ulteriori Basic Block in presenza di chiamate a funzione, in quanto l'istruzione stessa contiene al suo interno questa informazione.

2. CFG Loop

Similmente alla printf, la funzione g_incr non include variazioni di controllo di flusso.

```
; Function Attrs: nofree norecurse nounwind uwtable
define dso_local i32 @g_incr(i32 %0) local_unnamed_addr #0 {
  %2 = load i32, i32* @g, align 4, !tbaa !2
  %3 = add nsw i32 %2, %0
  store i32 %3, i32* @g, align 4, !tbaa !2
  ret i32 %3
}
```

3. Considerazioni

Si riporta prima il bytecode prodotto in modalità -O0, ovvero senza ottimizzazioni, e di seguito la corrispondente forma in modalità -O2 per la funzione di Loop.

```
; ModuleID = 'test/Loop.c'
source_filename = "test/Loop.c"
target datalayout = "e-m:e-p270:32:32-p271:32:32-p272:64:64-i64:64-f80:128-n8:16:32:64-S128"
target triple = "x86_64-pc-linux-gnu"

@g = dso_local global i32 0, align 4

; Function Attrs: noinline nounwind optnone uwtable
define dso_local i32 @g_incr(i32 noundef %0) #0 {
  %2 = alloca i32, align 4
  store i32 %0, i32* %2, align 4
  %3 = load i32, i32* %2, align 4
  %4 = load i32, i32* @g, align 4
  %5 = add nsw i32 %4, %3
  store i32 %5, i32* @g, align 4
  %6 = load i32, i32* @g, align 4
  ret i32 %6
}

; Function Attrs: noinline nounwind optnone uwtable
define dso_local i32 @loop(i32 noundef %0, i32 noundef %1, i32 noundef %2) #0 {
  %4 = alloca i32, align 4
  %5 = alloca i32, align 4
  %6 = alloca i32, align 4
  %7 = alloca i32, align 4
  %8 = alloca i32, align 4
  store i32 %0, i32* %4, align 4
  store i32 %1, i32* %5, align 4
  store i32 %2, i32* %6, align 4
  store i32 0, i32* %8, align 4
  %9 = load i32, i32* %4, align 4
  store i32 %9, i32* %7, align 4
  br label %10

10:                                               ; preds = %17, %3
  %11 = load i32, i32* %7, align 4
  %12 = load i32, i32* %5, align 4
  %13 = icmp slt i32 %11, %12
  br i1 %13, label %14, label %20

14:                                               ; preds = %10
  %15 = load i32, i32* %6, align 4
  %16 = call i32 @g_incr(i32 noundef %15)
  br label %17

17:                                               ; preds = %14
  %18 = load i32, i32* %7, align 4
  %19 = add nsw i32 %18, 1
  store i32 %19, i32* %7, align 4
  br label %10, !llvm.loop !6

20:                                               ; preds = %10
  %21 = load i32, i32* %8, align 4
  %22 = load i32, i32* @g, align 4
  %23 = add nsw i32 %21, %22
  ret i32 %23
}
```

```
; ModuleID = 'test/Loop.c'
source_filename = "test/Loop.c"
target datalayout = "e-m:e-p270:32:32-p271:32:32-p272:64:64-i64:64-f80:128-n8:16:32:64-S128"
target triple = "x86_64-pc-linux-gnu"

@g = dso_local local_unnamed_addr global i32 0, align 4

; Function Attrs: mustprogress nofree norecurse nosync nounwind uwtable willreturn
define dso_local i32 @g_incr(i32 noundef %0) local_unnamed_addr #0 {
  %2 = load i32, i32* @g, align 4, !tbaa !5
  %3 = add nsw i32 %2, %0
  store i32 %3, i32* @g, align 4, !tbaa !5
  ret i32 %3
}

; Function Attrs: nofree norecurse nosync nounwind uwtable
define dso_local i32 @loop(i32 noundef %0, i32 noundef %1, i32 noundef %2) local_unnamed_addr #1 {
  %4 = load i32, i32* @g, align 4, !tbaa !5
  %5 = icmp sgt i32 %1, %0
  br i1 %5, label %6, label %10

6:                                                ; preds = %3
  %7 = sub i32 %1, %0
  %8 = mul i32 %7, %2
  %9 = add i32 %4, %8
  store i32 %9, i32* @g, align 4, !tbaa !5
  br label %10

10:                                               ; preds = %6, %3
  %11 = phi i32 [ %9, %6 ], [ %4, %3 ]
  ret i32 %11
}

attributes #0 = { mustprogress nofree norecurse nosync nounwind uwtable willreturn "frame-pointer"=
attributes #1 = { nofree norecurse nosync nounwind uwtable "frame-pointer"="none" "min-legal-vector

!llvm.module.flags = !{!0, !1, !2, !3}
!llvm.ident = !{!4}

!0 = !{i32 1, !"wchar_size", i32 4}
!1 = !{i32 7, !"PIC Level", i32 2}
!2 = !{i32 7, !"PIE Level", i32 2}
!3 = !{i32 7, !"uwtable", i32 1}
!4 = !{!"Ubuntu clang version 14.0.0-1ubuntu1"}
!5 = !{!6, !6, i64 0}
!6 = !{!"int", !7, i64 0}
!7 = !{!"omnipotent char", !8, i64 0}
!8 = !{!"Simple C/C++ TBAA"}
```

Si può notare innanzitutto che tutte le istruzioni alloca vengono eliminate, in quanto non necessarie, dal pass di ottimizzazione “mem2reg”. Nella funzione loop, viene ridotto il numero di BasicBlock. Il loop presente all’interno della funzione viene sostituito con 3 semplici istruzioni: una sottrazione, una moltiplicazione, e un’addizione. Questo è stato possibile solamente in seguito all’inlining della funzione g_incr. L’inlining è tipicamente propedeutico a molte ottimizzazioni.

Eseguendo il seguente comando:
`clang -O2 -emit-llvm -S -c test/Loop.c -o test/Loop02.ll -Rpass=.*`
vengono stampate in dettaglio le ottimizzazioni che vengono effettuate.

```
test/Loop.c:34:5: remark: 'g_incr' inlined into 'loop' with (cost=-20, threshold=337) at callsite loop:4:5; [-Rpass=inline]
    g_incr(c);
    ^
test/Loop.c:21:5: remark: Moving accesses to memory location out of the loop [-Rpass=licm]
  g += c;
    ^
test/Loop.c:21:5: remark: Moving accesses to memory location out of the loop [-Rpass=licm]
test/Loop.c:33:3: remark: Loop deleted because it is invariant [-Rpass=loop-delete]
  for (i = a; i < b; i++) {
  ^
test/Loop.c:37:16: remark: load of type i32 eliminated [-Rpass=gvn]
  return ret + g;
               ^
```

Si prendono ora in considerazione le ottimizzazioni che sono state effettuate nella funzione fibonacci.

```
@stdout = external global %struct._IO_FILE*, align 8
@.str = private unnamed_addr constant [9 x i8] c"f(0) = 0\00", align 1
@.str.1 = private unnamed_addr constant [9 x i8] c"f(1) = 1\00", align 1
@.str.2 = private unnamed_addr constant [22 x i8] c"f(%d) = f(%d) + f(%d)\00", align 1

; Function Attrs: noinline nounwind optnone uwtable
define dso_local i32 @printf(i8* noundef %0, ...) #0 {
  %2 = alloca i8*, align 8
  %3 = alloca i32, align 4
  %4 = alloca [1 x %struct.__va_list_tag], align 16
  store i8* %0, i8** %2, align 8
  %5 = getelementptr inbounds [1 x %struct.__va_list_tag], [1 x %struct.__va_list_tag]* %4, i64 0, i64 0
  %6 = bitcast %struct.__va_list_tag* %5 to i8*
  call void @llvm.va_start(i8* %6)
  %7 = load %struct._IO_FILE*, %struct._IO_FILE** @stdout, align 8
  %8 = load i8*, i8** %2, align 8
  %9 = getelementptr inbounds [1 x %struct.__va_list_tag], [1 x %struct.__va_list_tag]* %4, i64 0, i64 0
  %10 = call i32 @vfprintf(%struct._IO_FILE* noundef %7, i8* noundef %8, %struct.__va_list_tag* noundef %9)
  store i32 %10, i32* %3, align 4
  %11 = getelementptr inbounds [1 x %struct.__va_list_tag], [1 x %struct.__va_list_tag]* %4, i64 0, i64 0
  %12 = bitcast %struct.__va_list_tag* %11 to i8*
  call void @llvm.va_end(i8* %12)
  %13 = load i32, i32* %3, align 4
  ret i32 %13
}

; Function Attrs: nofree nosync nounwind willreturn
declare void @llvm.va_start(i8*) #1

declare i32 @vfprintf(%struct._IO_FILE* noundef, i8* noundef, %struct.__va_list_tag* noundef) #2

; Function Attrs: nofree nosync nounwind willreturn
declare void @llvm.va_end(i8*) #1

; Function Attrs: noinline nounwind optnone uwtable
define dso_local i32 @Fibonacci(i32 noundef %0) #0 {
  %2 = alloca i32, align 4
  %3 = alloca i32, align 4
  store i32 %0, i32* %3, align 4
  %4 = load i32, i32* %3, align 4
  %5 = icmp eq i32 %4, 0
  br i1 %5, label %6, label %8

6:                                                ; preds = %1
  %7 = call i32 (i8*, ...) @printf(i8* noundef getelementptr inbounds ([9 x i8], [9 x i8]* @.str, i64 0, i64 0))
  store i32 0, i32* %2, align 4
  br label %27

8:                                                ; preds = %1
  %9 = load i32, i32* %3, align 4
  %10 = icmp eq i32 %9, 1
  br i1 %10, label %11, label %13

11:                                               ; preds = %8
  %12 = call i32 (i8*, ...) @printf(i8* noundef getelementptr inbounds ([9 x i8], [9 x i8]* @.str.1, i64 0, i64 0))
  store i32 1, i32* %2, align 4
  br label %27

13:                                               ; preds = %8
  %14 = load i32, i32* %3, align 4
  %15 = load i32, i32* %3, align 4
  %16 = sub nsw i32 %15, 1
  %17 = load i32, i32* %3, align 4
  %18 = sub nsw i32 %17, 2
  %19 = call i32 (i8*, ...) @printf(i8* noundef getelementptr inbounds ([22 x i8], [22 x i8]* @.str.2, i64 0, i64 0), i32 noundef %14, i32 noundef %16, i32 noundef %18)
  %20 = load i32, i32* %3, align 4
  %21 = sub nsw i32 %20, 1
  %22 = call i32 @Fibonacci(i32 noundef %21)
  %23 = load i32, i32* %3, align 4
  %24 = sub nsw i32 %23, 2
  %25 = call i32 @Fibonacci(i32 noundef %24)
  %26 = add nsw i32 %22, %25
  store i32 %26, i32* %2, align 4
  br label %27

27:                                               ; preds = %13, %11, %6
  %28 = load i32, i32* %2, align 4
  ret i32 %28
}
```

```
@stdout = external local_unnamed_addr global %struct._IO_FILE*, align 8
@.str = private unnamed_addr constant [9 x i8] c"f(0) = 0\00", align 1
@.str.1 = private unnamed_addr constant [9 x i8] c"f(1) = 1\00", align 1
@.str.2 = private unnamed_addr constant [22 x i8] c"f(%d) = f(%d) + f(%d)\00", align 1

; Function Attrs: nofree nounwind uwtable
define dso_local i32 @printf(i8* nocapture noundef readonly %0, ...) local_unnamed_addr #0 !dbg !8 {
  %2 = alloca [1 x %struct.__va_list_tag], align 16
  %3 = bitcast [1 x %struct.__va_list_tag]* %2 to i8*, !dbg !11
  call void @llvm.lifetime.start.p0i8(i64 24, i8* nonnull %3) #4, !dbg !11
  %4 = getelementptr inbounds [1 x %struct.__va_list_tag], [1 x %struct.__va_list_tag]* %2, i64 0, i64 0, !dbg !12
  call void @llvm.va_start(i8* nonnull %3), !dbg !12
  %5 = load %struct._IO_FILE*, %struct._IO_FILE** @stdout, align 8, !dbg !13, !tbaa !14
  %6 = call i32 @vfprintf(%struct._IO_FILE* noundef %5, i8* noundef %0, %struct.__va_list_tag* noundef nonnull %4), !dbg !18
  call void @llvm.va_end(i8* nonnull %3), !dbg !19
  call void @llvm.lifetime.end.p0i8(i64 24, i8* nonnull %3) #4, !dbg !20
  ret i32 %6, !dbg !21
}

; Function Attrs: argmemonly mustprogress nofree nosync nounwind willreturn
declare void @llvm.lifetime.start.p0i8(i64 immarg, i8* nocapture) #1

; Function Attrs: mustprogress nofree nosync nounwind willreturn
declare void @llvm.va_start(i8*) #2

; Function Attrs: nofree nounwind
declare noundef i32 @vfprintf(%struct._IO_FILE* nocapture noundef, i8* nocapture noundef readonly, %struct.__va_list_tag* noundef) local_unnamed_addr #3

; Function Attrs: mustprogress nofree nosync nounwind willreturn
declare void @llvm.va_end(i8*) #2

; Function Attrs: argmemonly mustprogress nofree nosync nounwind willreturn
declare void @llvm.lifetime.end.p0i8(i64 immarg, i8* nocapture) #1

; Function Attrs: nofree nounwind uwtable
define dso_local i32 @Fibonacci(i32 noundef %0) local_unnamed_addr #0 !dbg !22 {
  br label %2, !dbg !23

2:                                                ; preds = %5, %1
  %3 = phi i32 [ 0, %1 ], [ %10, %5 ]
  %4 = phi i32 [ %0, %1 ], [ %7, %5 ]
  switch i32 %4, label %5 [
    i32 0, label %12
    i32 1, label %11
  ], !dbg !24

5:                                                ; preds = %2
  %6 = add nsw i32 %4, -1, !dbg !25
  %7 = add nsw i32 %4, -2, !dbg !26
  %8 = tail call i32 (i8*, ...) @printf(i8* noundef nonnull dereferenceable(1) getelementptr inbounds ([22 x i8], [22 x i8]* @.str.2, i64 0, i64 0), i32 noundef %4, i32 noundef %6, i32 noundef %7), !dbg !27
  %9 = tail call i32 @Fibonacci(i32 noundef %6), !dbg !28
  %10 = add nsw i32 %9, %3, !dbg !29
  br label %2, !dbg !23

11:                                               ; preds = %2
  br label %12, !dbg !30

12:                                               ; preds = %2, %11
  %13 = phi i8* [ getelementptr inbounds ([9 x i8], [9 x i8]* @.str.1, i64 0, i64 0), %11 ], [ getelementptr inbounds ([9 x i8], [9 x i8]* @.str, i64 0, i64 0), %2 ]
  %14 = tail call i32 (i8*, ...) @printf(i8* noundef nonnull dereferenceable(1) %13), !dbg !30
  %15 = add nsw i32 %4, %3, !dbg !29
  ret i32 %15, !dbg !31
}
```

In questo caso un'ottimizzazione di fondamentale importanza è l'eliminazione di una chiamata ricorsiva, che porta a miglioramenti estremi dei tempi di esecuzione. Una delle due chiamate ricorsive (precisamente Fibonacci(n-2)) viene eliminata grazie all'utilizzo di un loop. Si può notare infine che il numero di BasicBlock è diminuito grazie all'utilizzo di istruzioni phi.

Ancora una volta, con l'opzione -Rpass=.* si possono visualizzare le varie ottimizzazioni che sono state effettuate.

```
test/Fibonacci.c:24:29: remark: transforming tail recursion into loop [-Rpass=tailcallelim]
  return Fibonacci(n - 1) + Fibonacci(n - 2);
                          ^
test/Fibonacci.c:24:29: remark: advising against unrolling the loop because it contains a call [-Rpass=TTI]
test/Fibonacci.c:24:29: remark: advising against unrolling the loop because it contains a call [-Rpass=TTI]
```